EFFICACY OF FREQUENT, SECURE, AUTOMATED TESTING ON STUDENT ACADEMIC PERFORMANCE

BY

TERENCE NIP

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Advisor:

Research Professor Elsa L. Gunter

Abstract

This thesis discusses the effects of frequent, secure automated testing on student performance in CS 421, a class titled *Programming Languages and Compilers* at the University of Illinois at Urbana-Champaign, and examines in-depth the tools utilized to enable such testing. In particular, it looks into the use of PrairieLearn and the Computer-based Testing Center as a testing platform and service to determine the ramifications of automated testing. It was found that students experienced up to a 12 percent increase in their final grade in Fall 2015 over students in Fall 2014. Additionally, it was shown that the use of custom-built tools on PrairieLearn in enabling testing of Programming Languages-related concepts did not adversely affect and in some cases, statistically significantly increased student grades.

To my parents, for their unwavering support in my academic endeavours.

Acknowledgments

I would like to thank Dr. Elsa L. Gunter for her guidance, support, and dedication. I am incredibly indebted to her for the opportunities she has given me and the wisdom she has imparted on me. This thesis would not be possible without her.

I would like to thank the Academic Office of the Computer Science Department at the University of Illinois at Urbana-Champaign for their support, assistance, and offerings of food to remind me to eat lunch every day.

I would like to thank Dr. Susan B. Older, my undergraduate advisor, for providing me with both the guidance and the opportunities to pursue my academic interests. Without her, I would have pursued neither Programming Languages and Formal Methods, nor education.

I would like to thank Christabel Osei-Bobie Sheldon for her continuing support, encouragement, and wisdom.

Finally, I would like to thank my parents - my mother, Wing Wah, and my father, Shui Wing - for their love, support, and especially their patience.

Table of Contents

List of Figures

Chapter 1

Introduction

A major side effect of increasing class sizes is the difficulty in assessing students in a fair and timely manner, including the time it takes from students submitting their completed assessments to returning a graded assessment back to them. Additional complications that arise include exam security (for example, when exams are split between two times or are held in a room that is insufficient for the number of students taking the exam). In an effort to reduce the magnitude of these issues, CS 421 decided to turn towards using secure automated student assessment as a mechanism to handle assessing students at scale.

The solution the course turned to was to use PrairieLearn in the Computer-based Testing Facility. PrairieLearn helps automate grading of student assessments, with the Computer-based Testing Facility handling proctoring. Additionally, the Computer-based Testing Facility allows for asynchronous testing within a prescribed one-week period. This permits students to come in and complete their given assessment for the amount of time they would have been given if the assessment were done in-class.

As a result of the decreased load on course staff in assessing students, the class was able not only to have its normal assessments (i.e. midterm and final examinations), but also assess students on how well they completed and understood their homework by testing them on it as well. Ultimately, we were able to provide greater test coverage on the material covered in class and on homework assignments. This thesis will demonstrate that not only is this new method of frequent, secure automated testing just as effective, if not more, than traditional testing, but that it is beneficial for students because of the implicit requirement for students to stay on top of material.

Chapter 2

Background

Over the past decade, Computer Science departments across the country have been experiencing a steady increase in the number of students entering each class. [7] This growing influx, in conjunction with arguably mismanaged resources, leads to increased class sizes (to the point at which they are unmanagable) and decreased efficiacy of instruction due to less targeted teaching. In particular, this thesis looks at the problems facing CS 421, *Programming Languages and Compilers*.

The introduction of multiple choice questions on exams in CS 421 came about after students began to complain about the amount of writing that needed to be done on exams. As a way to combat those complaints, questions which used to be writing intensive (such as writing out complete proofs) were converted to multiple choice questions. This had the obvious benefit of decreased writing time on the part of students during a timed exam, but at the cost of requiring more subtle thinking and a restricted ability to determine a broader scope of misunderstanding. That is, instead of being able to identify the chain of events which led to the misunderstanding, one is now only able to pinpoint the end result of the misunderstanding. As a side effect, the multiple choice questions decreased the workload on the course staff. By having to grade within a finite set of choices instead of a freeform solution with virtually infinite permutations which may or may not be correct, the amount of time and the number of mental cycles required to grade those questions both decreased significantly.

With the increase in class size also came an increase in the percentage of the students who understood what was happening in their weekly Machine Problems (MPs), problem sets given at the University of Illinois at Urbana-Champaign requiring significant amounts of programming. Given the number of students passing through CS 421 every semester, students began not only to

collaborate with each other (to the point where the academic integrity lines were beginning to blur), but also to look towards external resources for guidance on how to complete assignments. The use of PrairieLearn allowed the course to increase assessment integrity while retaining the same quality of grading and feedback, while leaving the students the freedom to seek other resources.

The standard measures taken in previous semesters in CS 421 did not scale well when class sizes start ballooning to the point where human grading is impossible. As such, there exists a need for secure automated student assessment platforms to handle facilitation of student assessments and partial automated grading. Our solution to combat both this problem and the issue of resolving students' detrimental approaches to learning was to turn to *PrairieLearn*, a platform developed by Professor Matthew West of the Mechanical Science and Engineering Department at the University of Illinois at Urbana-Champaign [5], and the use of the *Computer-based Testing Facility* (CBTF), a service offered by Engineering IT at the University of Illinois at Urbana-Champaign [2]. In particular, the use of their services allowed the course to increase assessment integrity while retaining the same (if not better) quality of grading and feedback.

Chapter 3

Engineering Work

3.1 Primary Goals

The primary goals CS 421 had in deciding to leverage PrairieLearn and the Computer-based Testing Facility as a mechanism to offer secure automated student testing were to:

1. Reliably assess students' ability to code
2. Assess students' ability to convert a program specification into code
3. Test students' understanding of the mathematical underpinnings of subjects within Programming Languages under tight time constraints
4. Provide an intuitive user interface for students to complete computations in a time-efficient manner

3.1.1 Machine Labs

As part of the course's initial foray into the realm of secure automated testing, we introduced Machine Labs (MLs) into the curriculum in an effort to have students demonstrate their knowledge of at least some portion of the problems given in the class. With an ML, students are given an in-depth, moderately complex programming assignment and a week to work on it at home with a test suite and a pre-compiled solution to help them perform self-assessment. They are then subsequently tested on portions of the given assignment in the CBTF. MLs not only implicitly helped the course enforce academic integrity, but also allowed us to target testing towards portions

of an assignment that are deemed to be mathematically interesting or sufficiently complex enough to consider completion as a proxy for indicating understanding of an assignment.

3.1.2 Assessing Student Ability to Identify Errors

Another problem that the class sought to resolve was the amount of writing given to students during a time-limited assessment while maintaining a mechanism to assess a student's understanding of the mathematical concepts at hand. Our approach in resolving this issue was to allow students to identify errors in computations that we would provide to them. For example, they would have to identify an inference in a proof tree which were incorrect by selecting it - effectively a multiple-choice question at its core (Figure A.1). This allows students to do less writing while still allowing the assessment to test how thoroughly they understood the material, but at the cost of not allowing students to both fully demonstrate their understanding and independently demonstrate basic core competencies of the topics at hand.

3.1.3 Student Computations

The last approach taken to assess students effectively was to allow students to do computations by themselves. This meant that the course had to provide a mechanism by which we would offer students the flexibility of doing computations on paper while providing them with the basic structure of computations in order to help speed students up.

There were two ways we went about doing this, the first of which was to reformulate questions (Figure A.2) to be multiple choice, giving students a choice between statements which may or may not be true to determine their validity. That meant students would still be forced to complete a full computation as they would on paper, but instead were asked whether or not a statement is true or not based on the computation they've done (Figure A.3). This achieves the goal of trying to maintain parity with previous exams and testing students' understanding of the underlying mathematics whilst simultaneously decreasing the work on graders, but at the cost of increased time spent by students in digesting each statement given to them to verify.

The second method we took to enable students to pursue computations on their own was to

allow them to use a customized editor to draw out full proof trees and parse trees. This allows us to have virtually one-to-one parity between the old paper exams given in the past and the new electronic exams currently used, with the only potential difference anecdotally being the amount of time now required to complete a question.

These two methods not only afford students the flexibility required to perform computations on par with those given on paper. Anecdotal evidence suggests that the use of the given customized editors decreased the time spent on computations because of the ability to copy and paste content and the decreased amount of detailed, legible writing required. However, there does exist overhead from learning how to use the custom editors.

3.2 Implementation

As part of the engineering work done to enable the course's effective use of PrairieLearn, I wrote code to enable CS 421 course staff to ask students more complex, Programming Languages-related questions. The ultimate goal of this was both to enable greater efficiency for students and course staff, and to maintain parity with the material asked on previous years' exams.

3.2.1 Existing PrairieLearn Tools

PrairieLearn was built to be a platform that is subject-independent. As such, the question types designed for it are generic in implementation. Initially, PrairieLearn had four question types: Multiple Choice, Checkbox, File Upload, and Calculation. These question types are built into the platform itself, offering additional functionality such as input validation.

In building the CS421-specific tools, I leveraged the Calculation question type as a method by which student submissions could be sent to the PrairieLearn database. In particular, by serializing the HTML form backing each question type into JSON and having it be the value of the text box given by the question, we are able to submit arbitrary student responses for question types we create independently.

3.2.2 Question Type: Multiple True/False

Initially, we wanted to ask students basic true/false questions, but we found PrairieLearn's default question types to be lacking in richness. As such, I built in functionality to ask students multiple true/false questions within one PrairieLearn question.

There were a number of design decisions made to help make this question type intuitive for students. First, we decided to use checkboxes in lieu of radio buttons. This affords students the implicit knowledge that they can uncheck a selected answer, whereas radio buttons do not provide this intuition. Second, in an effort to ensure that students would only be allowed to choose at most one of the two choices given, we built in mutual exclusion between each pair of true/false checkboxes so that checking one box would uncheck the other, if checked. This, in conjunction with an external autograder run during final grading that checks for edge cases (like blank answers or double-answers), serves to ensure the integrity of student responses to this question type.

3.2.3 Question Type: Multiple True/False with Text

The Multiple True/False question type, however, did not let us truly gauge how well students were understanding the material. Even with the introduction of guessing penalties (given for incorrect answers), it is not evident whether or not students have grasped the course content being presented. As such, we introduced a new question type allowing students to provide both a true/false answer and their rationale for that answer (Figure A.4). This method allows us to reward students for their correct understanding of the material while giving us the opportunity to see where student misunderstandings lie.

The grading, unlike other multiple choice questions, happens after all exams have been completed; that is, grading occurs only when course staff run an external autograder on top of the results.

3.2.4 Question Type: Dropdown Blank Completion

Another form of question we wished to ask were complex questions in a multiple choice-like format while being able to present the given choices in context with the problem at hand. As such, I developed a question type where, for an arbitrary number of dropdown boxes, a student would be able to make their choices below the presented question and have their choice be represented in the question itself (Figure A.5).

This not only allows students to think about the problem as a whole, but also maintains parity with what would be presented to students on a paper exam. Additionally given the multiple choice nature of the question, we are able to externally autograde the responses which reduces the time required for grading.

3.2.5 Question Type: Proof Trees

In an effort to ask more complex questions such as type derivations, natural semantics proofs, transition semantics proofs, and Floyd-Hoare logic proofs, I developed two libraries using HTML and JavaScript: one to display proof trees (and have students be able to identify errors in the proofs) (Figure A.1), and another to allow students to construct proof trees from scratch (Figures A.6, A.7, and A.8).

The HTML markup used to display the basic proof tree was shared between the two libraries; modifications were made as appropriate between the two libraries to enable functionality either to highlight certain portions of a proof as being incorrect or to allow students to construct their own proof trees.

3.2.6 Question Type: Parse Trees

Similar to the motivation for implementing proof trees, I wrote a library for students to "draw" parse trees in the browser so students could demonstrate mastery of grammars - in particular, recognizing when a grammar is ambiguous and how strings can be parsed given an unambiguous grammar (Figure A.9).

In this case, choices were made to restrict where students could place or move nodes in an effort to show them the appearance of a proper parse tree and to restrict the level of "incorrectness" that students attempt.

This tool, and the tool for proof trees, provide students with not only the basic structure of what a solution potentially looks like, but also with a canvas that is mostly analogous to what they would have if they were to take the exam on paper.

3.2.7 Feature: In-Browser Code Editing

To provide students with a slightly more intuitive user interface, I worked on integrating an in-browser code editor into our CS 421-specific set of PrairieLearn materials, whilst enabling others to leverage the code I've written by open sourcing a copy of the library, removing the CS 421-specific portions (Figure A.10). To enable this functionality, I leveraged the open-source ACE editor, a code editor written in JavaScript [1]. The PrairieLearn implementation of the editor is backed by a hidden text field, with the code being submitted verbatim as written by the student. This affords students an environment that looks and feels as if they are working in a real text editor (instead of uploading a file), decreasing the learning curve for students.

3.2.8 Question Content Development

All the previous contributions were made in an effort to enable us to ask more complex questions. Question development was a major part of what drove the engineering of these question types. The hours spent in discussions surrounding question development and the conclusions drawn from them ultimately led to the specification of the behavior of the question types.

3.2.9 Difficulties Encountered

There were quite a few difficulties that I encountered while working on these tools, the first being caching issues. Given the nature of the PrairieLearn platform being a single-page application, there were cases where JavaScript files used for our libraries were cached by the browser, resulting in the

display of incorrect options for some questions. This was resolved by writing a workaround such that each file would be loaded only once, with each question dictating the specific options required for its own instance of the relevant library versus delegating option loading to the (potentially cached) JavaScript library.

Another issue we ran into was that the design of the PrairieLearn platform initially meant that creating new questions required copying files from existing questions, potentially resulting in question content either being lost or there being extraneous content for each question. Additionally, the structure of the platform allows for inadvertent overriding of built-in functionality by including extraneous files. This was later partially resolved by the introduction of global client code in the PrairieLearn platform, where code could be freely used and shared between questions. However, the issue still persists in portions of the platform and is yet to be completely resolved [5].

Chapter 4

Analysis

4.1 Methodology of Testing

It has been shown that varied (and frequent) testing leads to higher grades overall - and in fact, leads to greater retention of material taught over time [8]. This was evidenced by the progress made by students in CS 421 between 2014 and 2015, with the primary difference being the introduction of MLs and PrairieLearn.

Figure A.11 plots the percentage of students who received a certain final grade or below for Fall of both 2014 and 2015 in CS 421. We can observe that for students who took CS 421 in Fall of 2014 and received a grade of 47 or above, if they were to have taken the course in Fall of 2015, they would have received a percentage boost of up to 12% (and on average, 8.24%). This is indicative of the benefits that frequent use of secure automated testing confers; that is, by making students more accountable for their work via frequent high integrity testing, students tend to perform better. We can also observe that students who fall below the 47 point mark have seemingly experienced little to no benefit from the use of frequent, secure automated testing, demonstrating that students who are bound to fail will fail.

Figure A.12 graphs the difference, for any given grade, in the percent of students who received that grade between 2014 and 2015. More concretely, it shows the behavior of the grades received in 2015 relative to that of 2014. From this, we can observe that between the two years, the difference for those up until the 47 point mark is minimal. However, once we reach roughly the 50 point mark, we can observe that the difference between both years begins to increase significantly, to peak at approximately a 32% difference. This is demonstrative of the fact that students in 2015

were performing better those in 2014 up until the 83 point mark, at which point the rate of benefit begins to decrease.

The increase in grades, however, could be associated to the Hawthorne effect, where when experiment subjects know they are being observed, they modify their behavior to be out of the norm [9]. In our case, this means that having introduced PrairieLearn and MLs into the course curriculum may have simply led the students to perform better because of the novelty of the changes made.

4.2 Efficacy of PrairieLearn/CBTF Tools

Ultimately, it is the case that the developed tools should help enhance the student learning experience by both making it easier for students to work on assessments and helping students perform equally as well on them, if not better. The first primary change made was to have students do multiple choice problems instead of full computations.

Figures A.13 and A.14 compare performance of multiple choice questions in 2014 to proof tree questions in 2015. In particular, A.13 looks at a question on Hoare Logic proofs, and A.14 looks at a question on Natural Semantics derivations. After considering the relative difficulty of the types of questions, it appears to be the case that having students do proof trees on PrairieLearn leads to students performing better than if students are to complete multiple choice questions on PrairieLearn. This is evidenced by the graph of 2015 being below that of 2014; more concretely, it means that the distribution of students who received a certain number of points has been shifted to the right (i.e., have received higher grades).

This can be attributed to the fact that multiple choice questions, when written properly, require greater thinking on the student's part. Students have to effectively grade the question's choices for their correctness, which students find to be much more difficult task than if they were to answer the question directly. When students are doing a full computation on their own, they are able to acutely keep track of where they are in a particular computation, potentially decreasing the number of mistakes made.

Looking at full computations, it seems to be the case that the shift from paper exams to secure, electronic exams on PrairieLearn leads to little change grade-wise. This can be seen in Figure A.15, which plots the percentage of students receiving a grade or below on a polymorphic type inferencing problem. In particular, observe that there are fewer students who fell in the 6 to 12 point range in 2014 than there were in 2015, and that more students fell in the 12 to 17 point range in 2015 than in 2014. This suggests that students in the middle struggled a little more from using the tool, whereas students in the high range were not hindered by the tool. We can also observe that students overall tend to perform better when having to use the proof tree tool, with a larger percentage of students falling into the average grade range.

Finally, the CBTF allows courses to install custom software on their machines. As such, the course provides a compiler to students for their use during exams. In theory, students would not only use the compiler to test if their code compilers, but also leverage the given (complete) test suites to check their solutions for functional correctness. Figures A.16 and A.17 plot the percentage of students receiving a particular grade for basic coding questions and a moderately complex coding question, respectively. From them, we can observe that providing a compiler has little to no effect on student grades. This can be attributed to the fact that students are given time to prepare with the materials they are being tested on, and the assessment is a reflection on their level of understanding of the material. That is, their grade is indicative of how well they understood the relevant course content, which is independent of the means by which the assessment is given.

Chapter 5

A Critique of PrairieLearn/CBTF

5.1 The Student Perspective

From the student's perspective, there are quite a few positives coming from the use of PrairieLearn and the CBTF. First, the ability for students to complete tests asynchronously allows students to pick an optimal time slot to complete their exam. That is, students are no longer forced to complete assessments potentially at a time that is suboptimal for them (e.g., if they aren't morning people). This lets students pick times that are best for them, allowing them to take their assessments at times that they feel allow them to succeed.

Another benefit from the use of PrairieLearn and the CBTF is that they are given a computer to use. In particular, they are also given a compiler, allowing them to properly test their code. As shown earlier in this thesis, it is the case that despite giving students a compiler, students perform the same on assessments. Anecdotally however, students feel more comfortable when in front of a computer with a compiler.

Finally, students are able to get a rough sense of how they performed on any given assessment because of PrairieLearn's instant feedback functionality. With exception of the free text and code that needs to be externally autograded or manually graded by a human, students are able to get a relatively close estimate of what their grade is, reducing the stress that comes with waiting for a final rendered grade.

5.2 The Instructor Perspective

One major issue that PrairieLearn possesses is the lack of easy extensibility. It is normally very difficult and cumbersome for instructors to ask more complex questions beyond multiple choice or free response. Instructors do not have the time to develop course materials in great depth from scratch; PrairieLearn has been touted as a platform that requires a large investment up-front, but one that will pay dividends down the line when a sufficiently large question bank has been created. This, however, means that instead of spending time developing new course materials, time is being spent rehashing old materials, trying to get them to fit into the PrairieLearn framework by rephrasing test questions, creating new exams, and spending many hours on learning the platform in such great depth that one can actually develop new question types riding on top of existing PrairieLearn infrastructure.

Another issue that plagues PrairieLearn is the change in how instructors perceive how a student performed on an assessment. With paper exams, one can easily get a feel of how students are performing as a whole because the whole exam is right in front of you; you can flip through an exam to get a more holistic view of how students are doing/how well students are understanding related concepts. With PrairieLearn and other similar systems, that implicit understanding is removed, and is in fact, replaced with a quantified measurement of how certain questions did relative to another. This leaves the instructor with a measurement of how students performed, but with no real sense of how well students are understanding the material at hand beyond that. In particular, instructors get a view of how students performed particular questions instead of a view of how particular students performed on an exam.

However, there do exist benefits for PrairieLearn. For example, PrairieLearn being open source is a great boon; it enables individuals from around the world to contribute code to the project. However, the fact that PrairieLearn is still a rather top-down project run by a few select individuals means that many decisions are made without consulting other stakeholders (i.e., instructors). That, in itself, has quite a few ramifications.

First, it means that instructors are at the project's behest when there are changes made to

the platform. For example, changes have been made in the way exams need to be written. The temporal investment made initially to get into PrairieLearn only seems to grow larger, although with the development of sufficient automation, one can work towards minimizing this investment. (One should note, however, that it means instructors have to invest more time either from themselves or their course staff in developing said automation.)

It also means that quite a bit of core functionality needs to be built outside of PrairieLearn. For example, support for autograders (a mechanism by which student responses to programming questions can be automatically graded) in PrairieLearn is being worked on [5]. This functionality only came about because of great demand coming from the Computer Science department; instead of PrairieLearn being a one-stop shop, instructors were using PrairieLearn as a student response repository. They would then run external autograders on top of student submissions after the fact. The dearth of advanced functionality, in conjunction with the learning curve that inherently comes with learning any platform and the few hours that instructors and course staff have to develop content, makes for a rather hard question: does one try to fit existing pieces into PrairieLearn and other similar platforms, or does one merely use PrairieLearn as a way to capture student responses, then use existing materials on top of the captured responses?

But perhaps the greatest issue of all is "*Quis custodiet ipsos custodes?*" With an online platform, there is no "hard copy" that one can point to and say definitively that the work submitted is in fact what the student submitted. Trust in the results given back to students is ultimately based on the trust that students and instructors have in the system. Given any dispute of the data collected by any online platform, the only record that exists is what was ultimately recorded in the database backing the platform; there is no hard evidence that a student actually submitted something, insufficient evidence that grades were not subject to bugs existing in the code backing exams, etc.

It is also important to note that paper exams confer one additional benefit that PrairieLearn and other platforms can't ever offer: the promise of 100% uptime. The inherent nature of the Internet and computer-based testing means that it is susceptible to both hardware and software issues, ranging from networking to actual platform outages which can potentially (and have in the

past) affected student exams. With paper exams, there is little risk involved once the exams are printed; the few issues that could crop up afterwards are easily resolvable by running additional copies of the exam or making corrections to it. With electronic exams, there exists greater room for error and failure, ranging from equipment malfunctions to user error. It is also worth noting that with paper exams, data is often easily recoverable if need be, whereas that is not necessarily the case with electronic exams (e.g., if a user's computer shuts down prior to saving).

The novelty of platforms like PrairieLearn has opened up quite a few buckets of worms, but given an increase of the number of students in classes (from 100, 200, to upwards of 800), these platforms are a solution that helps to alleviate the larger problem at hand.

Chapter 6

Related Work

There have been numerous atttempts at electronic student assessment systems, each with their own benefits and drawbacks. Many of them are geared towards the "status quo": multiple choice, fill in the blank, and free response questions. However, this model clearly does not work for courses which require greater specificity/more in-depth questioning to ensure that students have both learned and retained material properly, instead of learning towards an exam.

6.1 Blackboard

Blackboard is a closed-source product which is generally known for being a course-management system [3]. However, it does contain tools both for hosting quizzes and exams and for homework assignment submission. Their testing framework is one of those that caters to the bare minimum for qualifying as testing; one cannot go beyond having basic multiple choice/fill in the blank/free response questions. Despite that, Blackboard does provide relatively easy plug-and-play functionality between their testing framework and their gradebook framework, where grades are inserted with minimal effort from an assessment to the gradebook.

6.2 RELATE

RELATE, a framework developed by Professor Andreas Kloeckner at the University of Illinois at Urbana-Champaign, was developed at the same time as PrairieLearn was but with a different purpose in mind: it was built more towards testing within the realm of Computer Science, supporting

Python autograding/coding questions out of the box, in addition with the basic multiple choice/free response question types that PrairieLearn supports as well [6].

6.3 WebAssign

WebAssign is perhaps the most mature platform of the ones presented thus far in terms of computer-based testing; WebAssign allows instructors to assign homework questions to students based on specific textbooks, reducing the instructor's workload down to an initial investment in deciding what questions to ask students [4].

An inherent weakness in WebAssign is that there is no built-in functionality for student testing (which these other platforms support), but in lacking this functionality, allows it to focus on its strength: making it simple for instructors to assign homework problems (with dynamic content). It is worth noting that one can theoretically leverage WebAssign as a testing tool, but it is not designed to be one.

Another weakness is that WebAssign is closed-source and does not enable instructors to create their own content. The problems that can be given to students are prescribed by WebAssign, meaning that instructors are at their behest when it comes to trying to understand how well students are understanding content. Additionally, WebAssign only exists for specific subjects, and within those subjects, for specific textbooks - meaning that users of WebAssign are forced into using the materials WebAssign supports.

On the other hand, generic exam development is a hard problem to solve, seeing as instructors have different ways they want to test students; some instructors care just about the final answer, others want to see complete work, and others want to ask more complex questions than can be represented by typing text into a text box. By avoiding this problem entirely, WebAssign avoids not only diluting its product and the goodwill it's developed from instructors for it, but also avoids having to tackle this tough problem in a way that is acceptable for all parties involved.

Chapter 7

Future Work

Future work that could be done to further existing work done in the arena of secure automated student assessment is to develop tools that allow for greater assessment of calculations. By building tools that allow students to demonstrate basic competencies in topics, we can then gather more granular information on what students understand, and to what degree students understand the topics at hand. As a long term goal, we would like not only to be able to collect and assess every step of student work given, but do it in a virtually immediate fashion. Providing incremental feedback affords students an opportunity to learn from their mistakes as soon as they make them, decreasing both the amount of time it potentially takes to resolve a misunderstanding and the amount of time students possess a misconception of the course material.

Chapter 8

Conclusion

Ultimately, the work done as part of this thesis was geared towards developing a method by which courses like CS 421 could assess students' ability to take program specifications and convert them to code, test their understanding of the mathematical concepts backing Programming Languages, and make the given method as intuitive as possible. In working towards secure automated testing, the course has not only introduced new assessments, but also took steps towards making assessments more student-friendly while simultaneously decreasing the workload on course staff.

With Machine Labs, the course was not only able get a greater sense of how well students understood the mathematical underpinnings of an assessment, but also tangentially reinforced the notion that students need to keep up with work in the course. This is a change from the pre-PrairieLearn era, where students would either not complete assessments or collaborate a bit too much with their friends.

With the introduction of PrairieLearn assessments came a need to design questions that not only worked to assess how well students understood course material, but also provided close parity to paper exams. Through the implementation of various tools and additional functionality on top of what PrairieLearn provides, the course was able not only to achieve close parity, but also work towards increasing student grades. In fact, it was the case that students who completed PrairieLearn assessments experienced up to a 12% increase in their final grade, which is equivalent to approximately a whole letter grade.

These results, however, are subject to the Hawthorne effect; that is, they could simply be attributed to the fact that the course introduced new changes to the way assessments were being handled relative to that of previous semesters. There is, however, no way to ethically rule out the

Hawthorne effect; the only method by which one could rule out the Hawthorne effect would be to do actual trials during the semester, which is rather unethical. The closest we can get to the gold standard and definitively demonstrate the positive effects of frequent, secure automated testing is pure correlation - and given sufficient data (i.e., multiple years' worth of data), one may be able to demonstrate stronger correlation than is currently possible.

At the end of the day, the fact that the results demonstrate a positive effect on student grades is reassuring - and is, in fact, indicative of the notion that the course is heading in the right direction by assessing students more frequently in an environment that is familiar to them and making them accountable for their work.

Appendix A

Figures

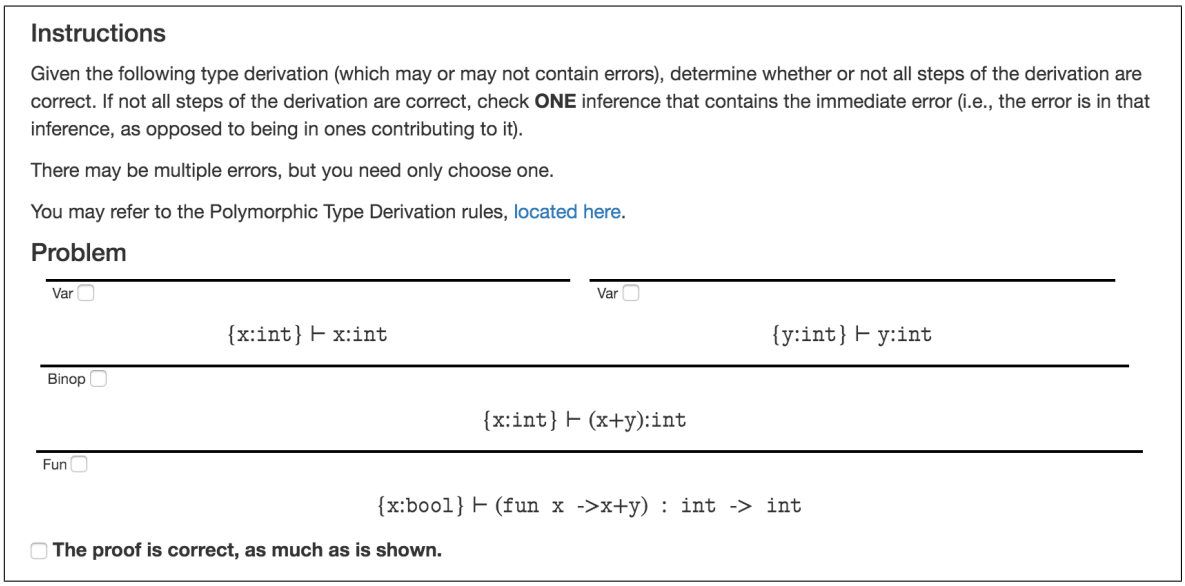

Figure A.1: A proof tree with checkboxes for identifying mistakes

Problem 7. (17 points)
Give a polymorphic type derivation for the following type judgment:

```
{ } |- let f = fun x -> x = x in f (f 3) : bool
```

Figure A.2: A sample proof tree question

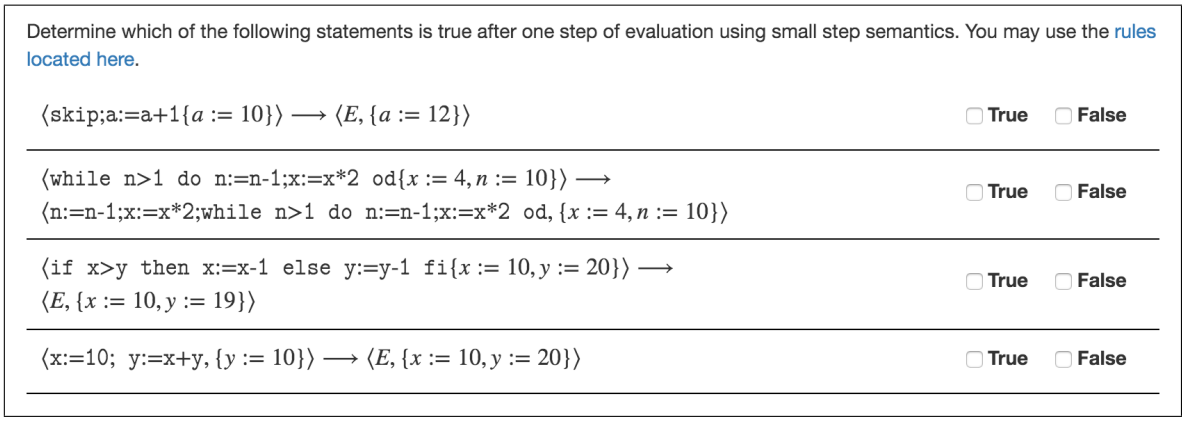

Figure A.3: A multiple true/false question

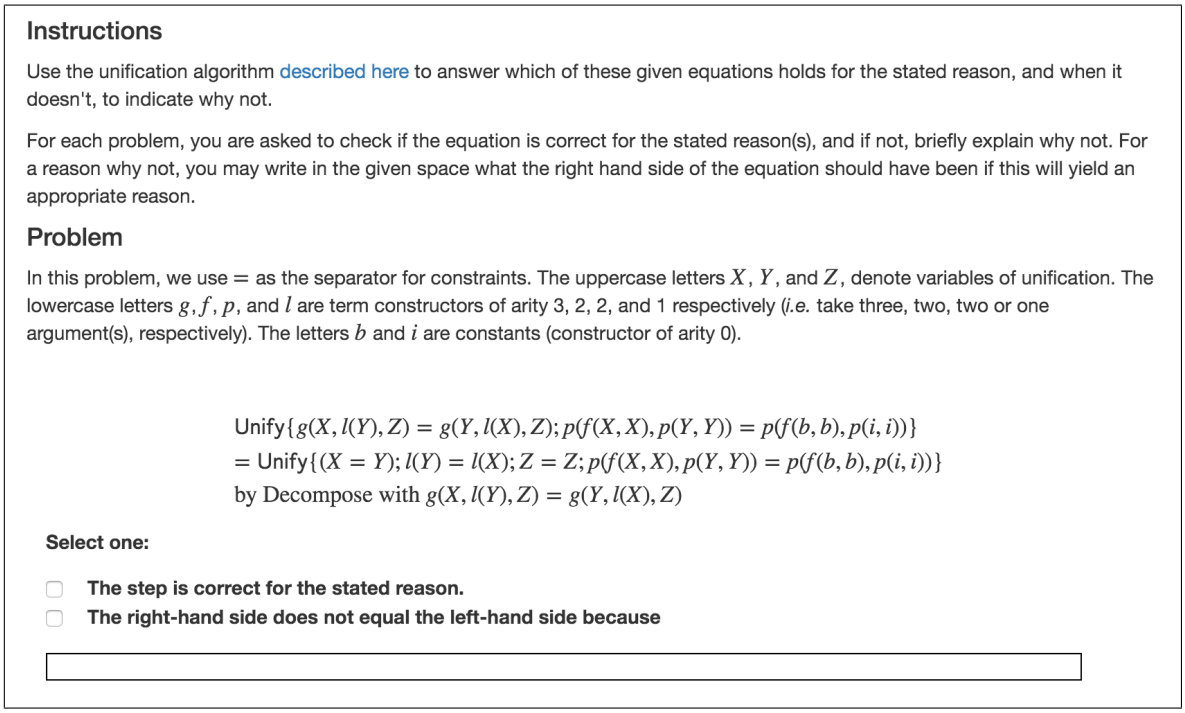

Figure A.4: A multiple true/false question with a text field

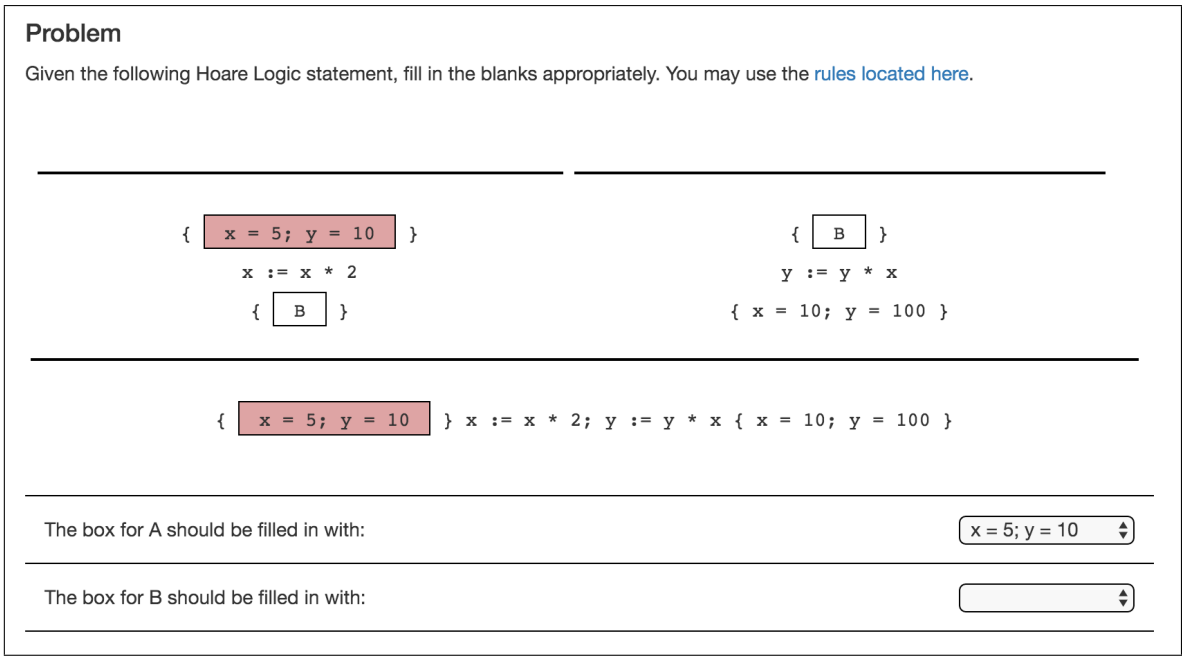

Figure A.5: A dropdown blank question

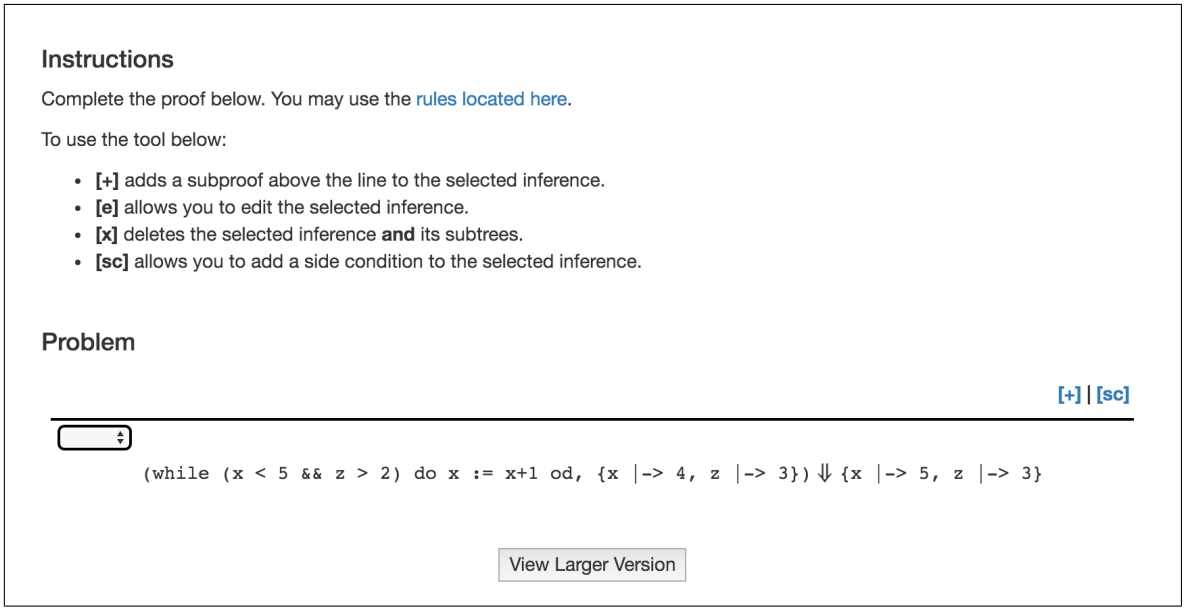

Figure A.6: A proof tree question

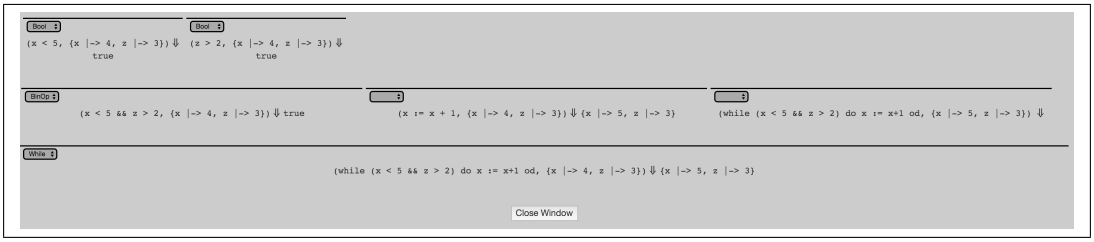

Figure A.7: A partial proof tree derivation in large view

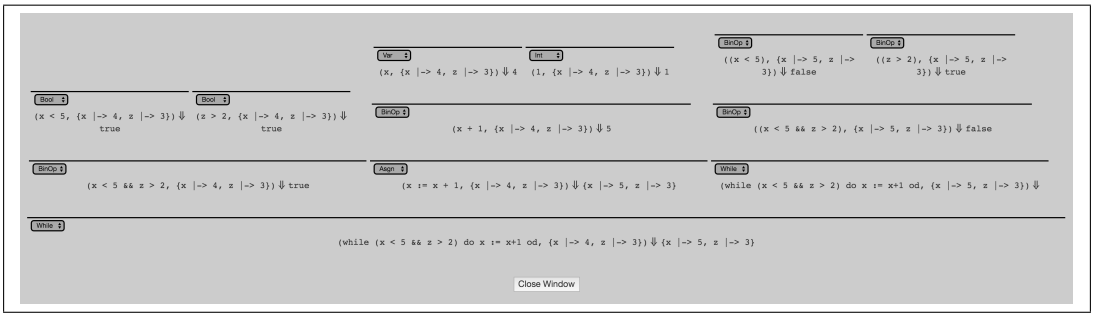

Figure A.8: A complete proof tree derivation in large view

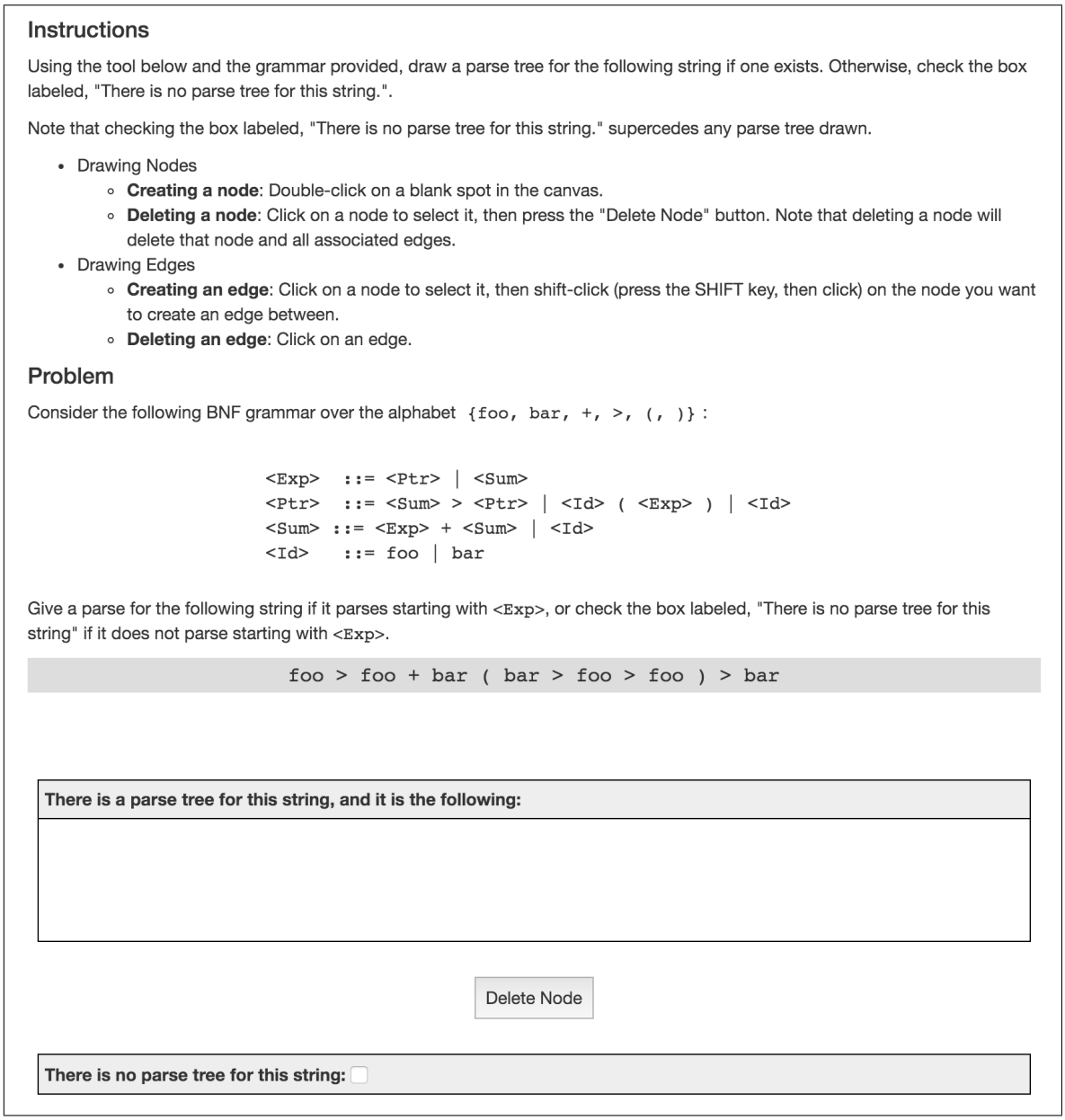

Figure A.9: A parse tree question

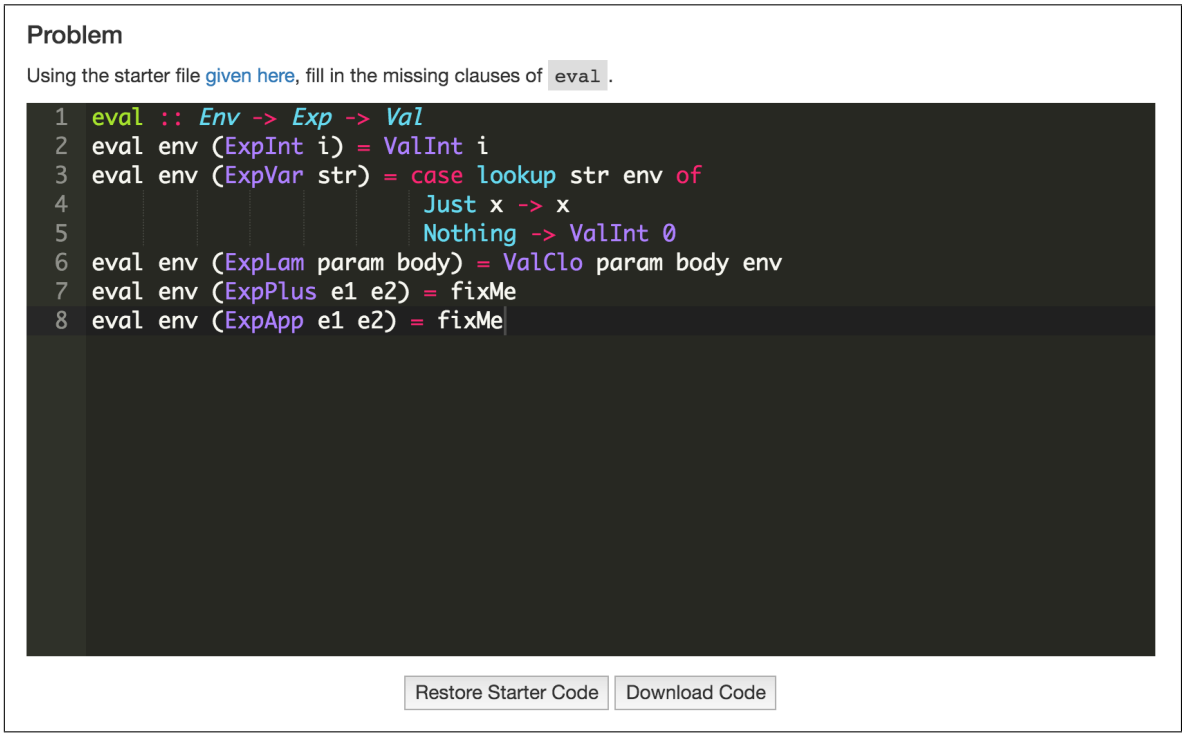

Figure A.10: A sample coding question

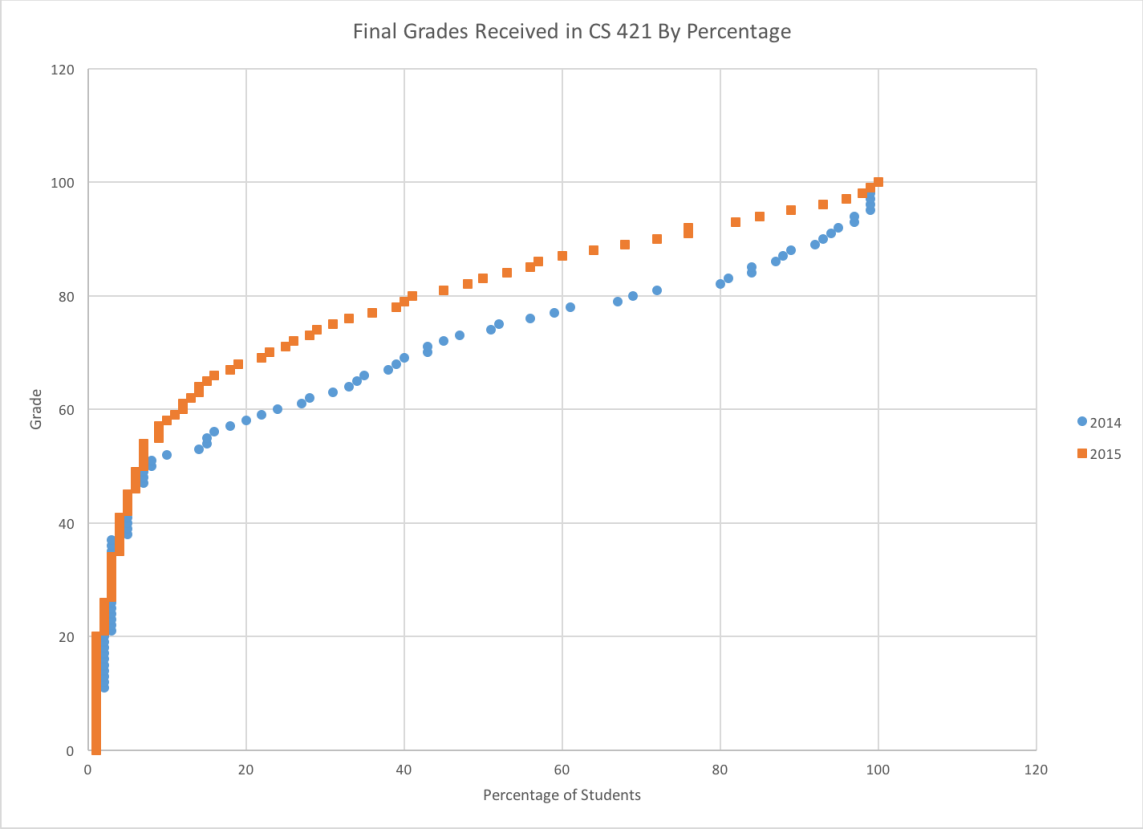

Figure A.11: Percent at or below final grade between 2014 and 2015

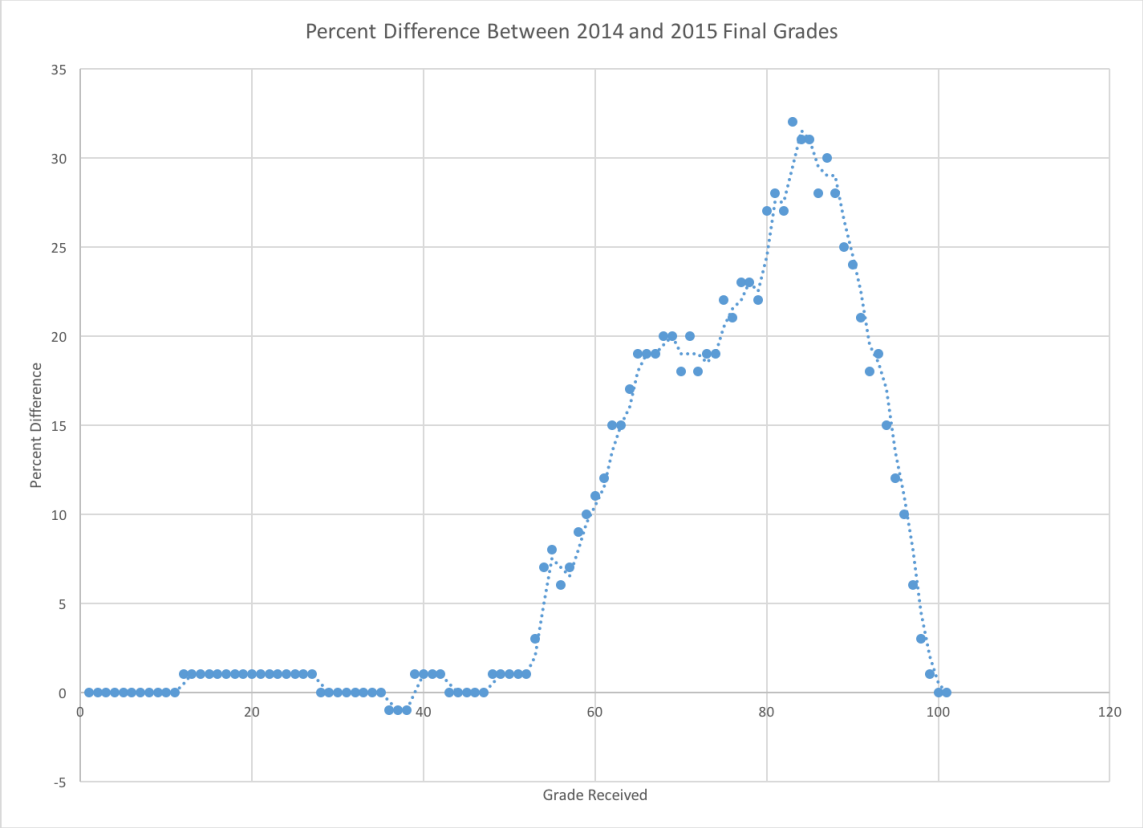

Figure A.12: Percent difference of students at or below final grade between 2014 and 2015

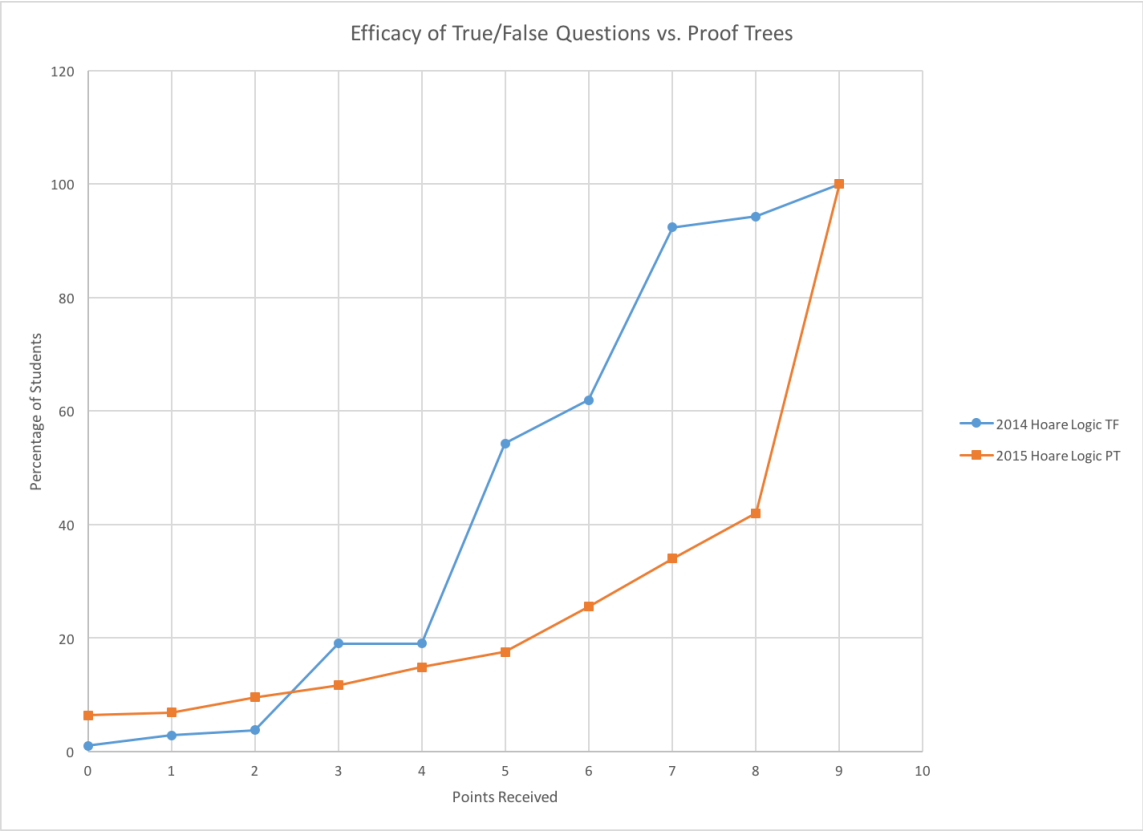

Figure A.13: Percent at or below grade for Hoare Logic problems (2014/2015 Final Exam)

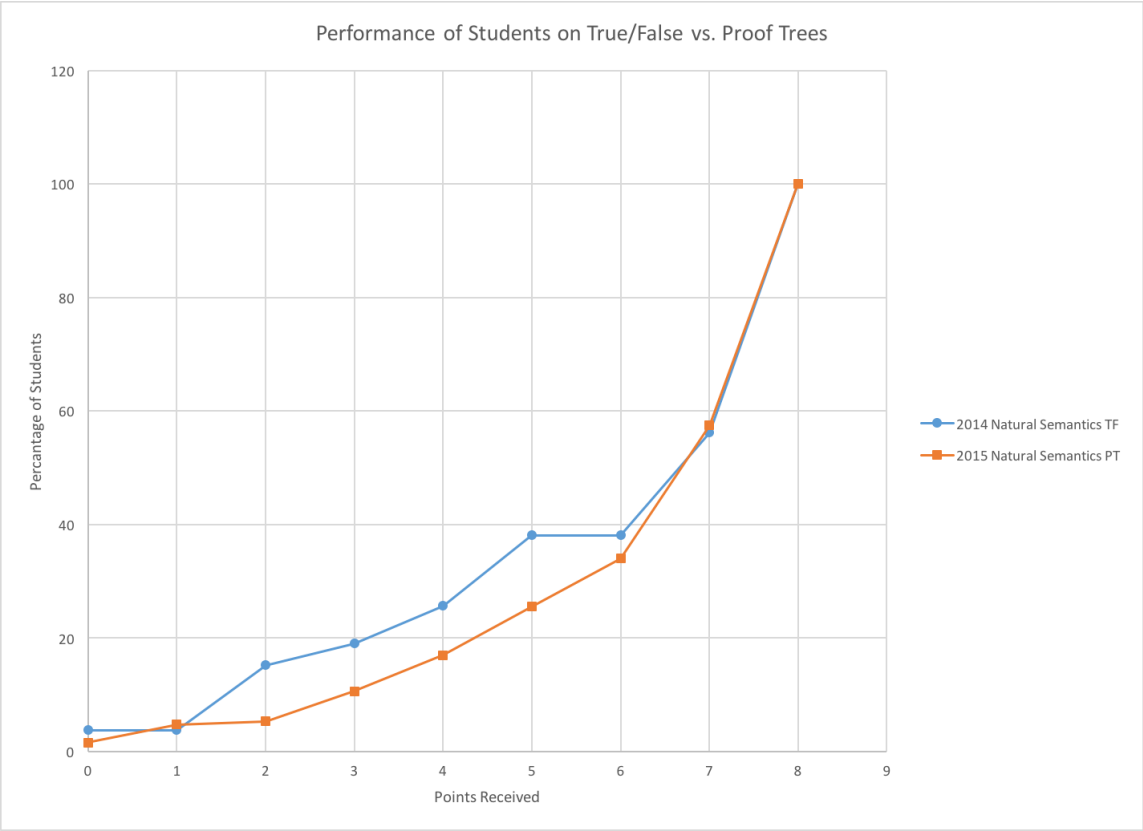

Figure A.14: Percent at or below grade for Natural Semantics problems (2014/2015 Final Exam)

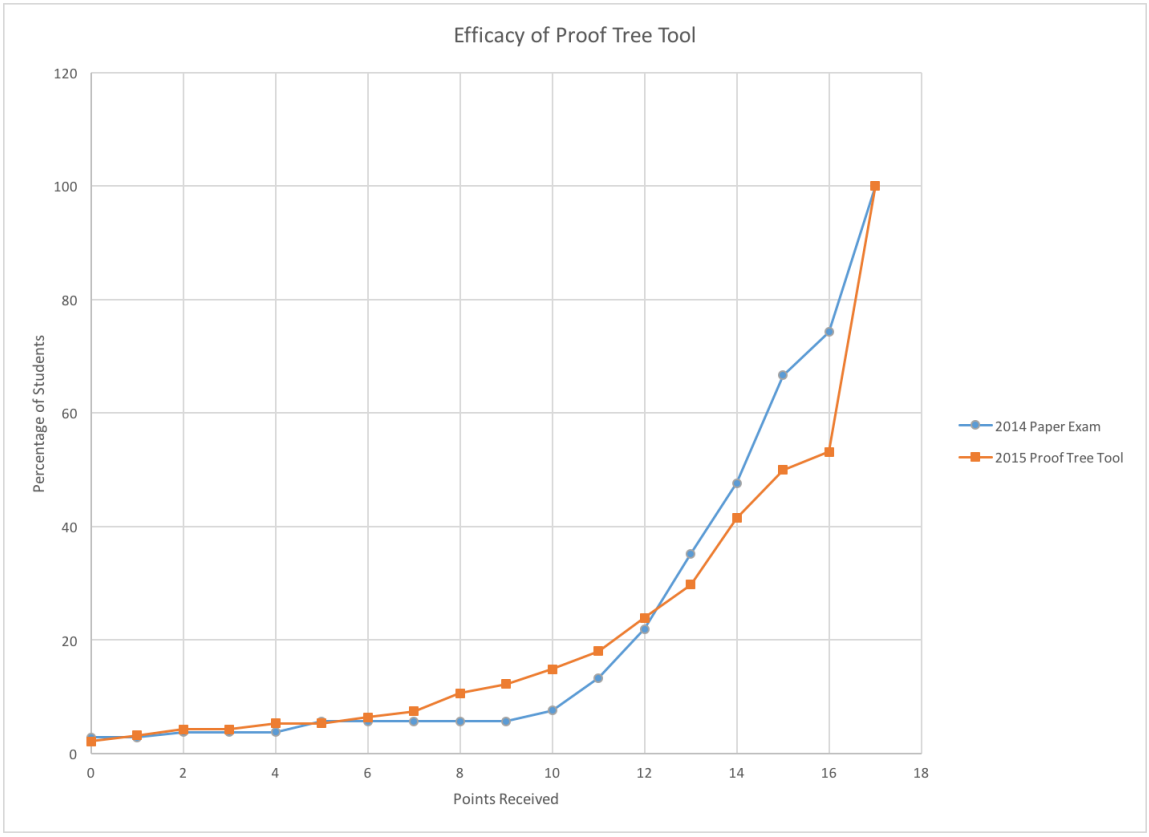

Figure A.15: Percent at or below grade for type derivation problems (2014/2015 Final Exam)

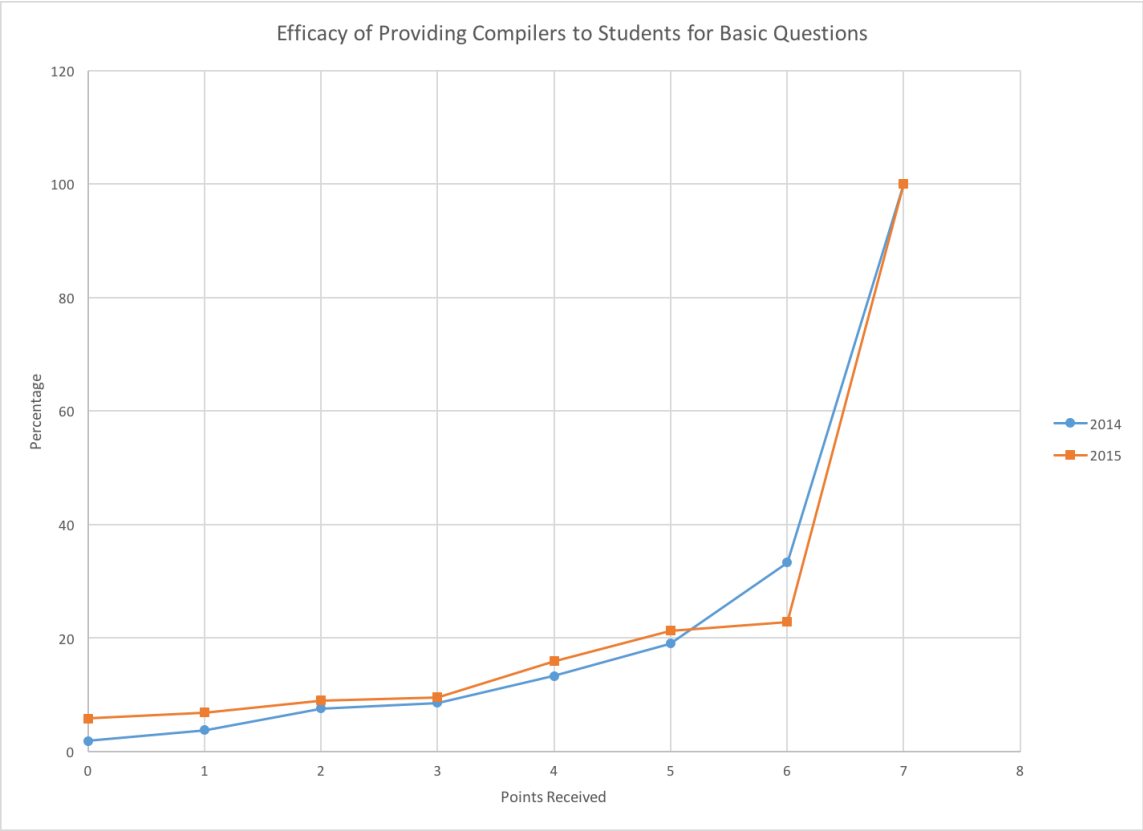

Figure A.16: Percent at or below grade for basic OCaml (2014/2015 Midterm 1)

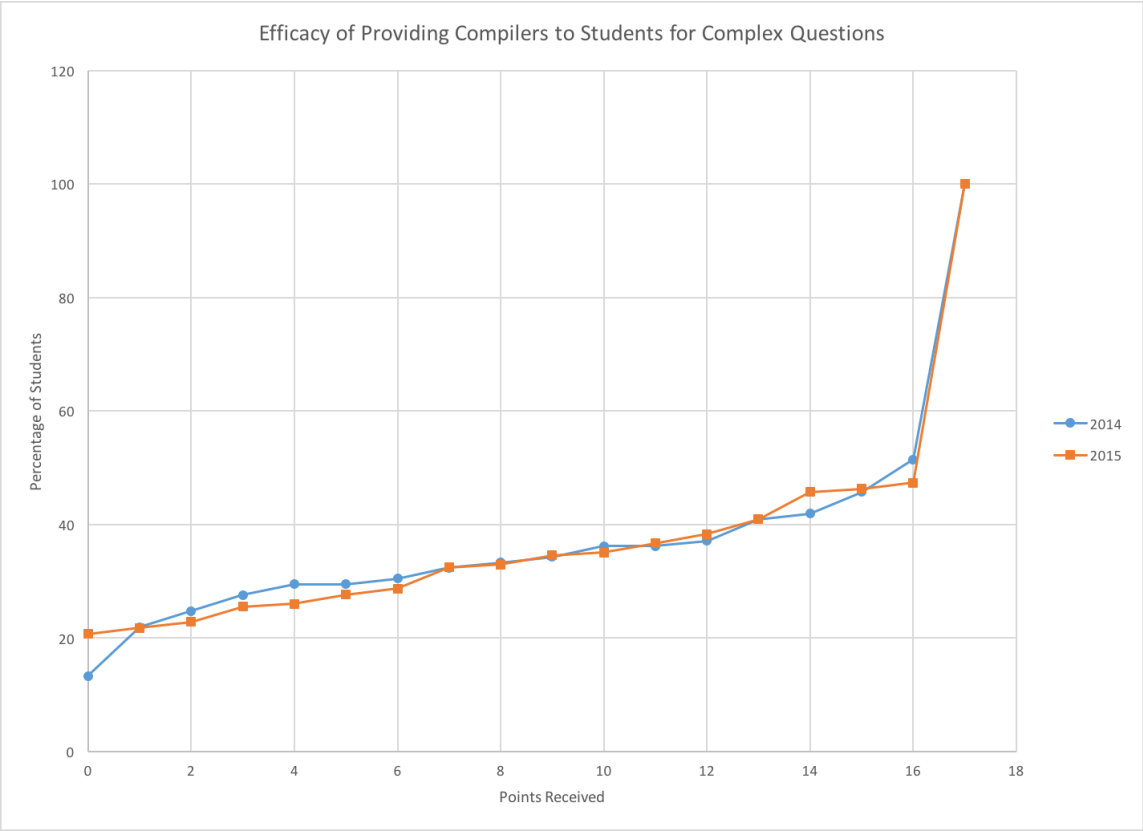

Figure A.17: Percent at or below grade for coding question (2014/2015 Final Exam)

Appendix B

Libraries & Question Samples

A sample of the CS 421-specific libraries and the questions using them may be found in a supplemental file named `code.zip`. Upon decompressing the ZIP archive, the following files will be available:

- `clientCode`
  - `ace-editor-helper.js`: The CS 421-specific Ace Editor library
  - `dropdowns-helper.js`: The library enabling the Dropdown Blank question type
  - `prooftree-draw-helper.js`: The library enabling the display of proof trees
- `clientFiles`
  - `css`: Contains style files for files in `html`
  - `html`: Contains documentation files for Inference and Unification Rules
  - `js`: Contains JavaScript files supporting files in `html`
- `questions`
  - `code-hs-adtBinaryTree-add`: A sample coding question with in-browser code editing
  - `dd-hoareLogic-01`: A sample drop-down blank completion problem
  - `final-lambdaCalc-2014a`: A sample multiple true/false problem
  - `final-parseTrees-elg1a`: A sample parse tree problem
  - `midterm2-polyTyDeriv-elgA01`: A sample proof tree question for identifying errors

– `midterm2-unification-elg`: A sample multiple true/false problem with text

– `prooftree-bigstep-bay`: A sample proof tree question

References

[1] Ace - The high performance code editor for the web. https://ace.c9.io/. Accessed: 2016-04-24.

[2] Computer-based testing facility. https://edu.cs.illinois.edu/cbtf/. Accessed: 2016-04-24.

[3] Create tests and surveys. https://en-us.help.blackboard.com/Learn/9.1_2014_04/Instructor/110_Tests_Surveys_Pools/010_Create_Tests_and_Surveys. Accessed: 2016-04-24.

[4] Features. https://webassign.com/instructors/features/. Accessed: 2016-04-24.

[5] Prairielearn. https://github.com/PrairieLearn/PrairieLearn. Accessed: 2016-04-24.

[6] Relate. https://github.com/inducer/relate. Accessed: 2016-04-24.

[7] E. Lazowska, E. Roberts, and E. Kurose. Tsunami or sea change? Responding to the explosion of student interest in computer science. https://www.ncwit.org/sites/default/files/edlazowska_ws_lowres_0.pdf. Accessed: 2016-04-24.

[8] James W. Pennebaker, Samuel D. Gosling, and Jason D. Ferrell. Daily online testing in large classes: Boosting college performance while reducing achievement gaps. *PLoS ONE*, 8(11):1–6, 11 2013.

[9] Harvard Business School. The "Hawthorne effect". http://www.library.hbs.edu/hc/hawthorne/09.html#nine. Accessed: 2016-04-24.